**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Расчетно-графическая работа

По дисциплине «Программирование»

 “Оценка трудоемкости сортировки”.

Группа: АВТ-041

Студент: Дударь М.А.

Преподаватель: Балакин В.В.

НОВОСИБИРСК 2021

1. **Постановка задачи:**

Реализовать программу сортировки одномерного массива пирамидальной сортировкой. В качестве исходных данных использовать случайные. Измерить «грязное» время выполнения программы и трудоемкость по основным операциям для различных значений.

1. **Алгоритм решения:**
2. Реализовать сортировку пирамидальным способом на языке C.
3. Собрать данные на разных размерностях одномерных массивов.
4. Произвести анализ полученных данных, используя язык Python 3.
5. **Принцип работы:**

Функции пирамидальной сортировки.

void heapify(int arr[], int n, int i, int \*rec\_count, int \*swap\_count) {  
 int largest = i;  
 (\*rec\_count)++;  
 int l = 2 \* i + 1;  
 int r = 2 \* i + 2;  
  
 if (l < n && arr[l] > arr[largest])  
 largest = l;  
  
 if (r < n && arr[r] > arr[largest])  
 largest = r;  
  
 if (largest != i) {  
 (\*swap\_count)++;  
 swap(&arr[i], &arr[largest]);  
  
 heapify(arr, n, largest, rec\_count, swap\_count);  
 }  
}  
  
void heapSort(int arr[], int n, int \*rec\_count, int \*swap\_count) {  
 for (int i = n / 2 - 1; i >= 0; i--)  
 heapify(arr, n, i, rec\_count, swap\_count);  
  
 for (int i = n - 1; i >= 0; i--) {  
 swap(&arr[0], &arr[i]);  
 (\*swap\_count)++;  
 heapify(arr, i, 0, rec\_count, swap\_count);  
 }  
}

Функции вызова сортировки на одномерных массивах разных размерностей и записи результатов в файлы.

void call(int len) {  
 int \*arr;  
 int rec\_count = 0, swap\_count = 0;  
 long time\_start, time\_finish;  
 arr = (int \*) malloc(len \* sizeof(int));  
 if (!arr)exit(-1);  
 FILE \*fp1;  
 FILE \*fp2;  
 if ((fp1 = fopen("write.txt", "a")) == NULL) {  
 printf("Can't open file 'write.txt'\n");  
 exit(-1);  
 }  
 if ((fp2 = fopen("write\_without\_time.txt", "a")) == NULL) {  
 printf("Can't open file 'write.txt'\n");  
 exit(-1);  
 }  
 for (int i = 0; i < len; ++i) {  
 arr[i] = rand() % RAND\_LIMITS;  
 }  
  
 time\_start = clock();  
 heapSort(arr, len, &rec\_count, &swap\_count);  
 time\_finish = clock();  
  
 fprintf(fp1, "%d %d %d %li\n", len, rec\_count, swap\_count, time\_finish - time\_start);  
 fprintf(fp2, "%d %d\n", len, rec\_count + swap\_count);  
 fclose(fp1);  
 fclose(fp2);  
 free(arr);  
}  
  
void call\_cycle(int len) {  
 for (int i = 0; i < CALL\_COUNT; ++i) {  
 call(len);  
 }  
}

1. **Результаты измерений:**

Теоретическая трудоемкость пирамидальной сортировки . Используя средства языка Python и библиотеки SciPy, учитывая, что (O) асимптотический множитель, вычислим этот (O) для данной выборки.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Теория |  | Практика | Отклонение |
| 50000 | 540988 | 1514768 | 1498503 | 1% |
| 70000 | 780937 | 2186625 | 2163937 | 1% |
| 90000 | 1026680 | 2874706 | 2850301 | 0.8% |
| 100000 | 1151292 | 3223619 | 3198107 | 0.79% |
| 150000 | 1787758 | 5005724 | 4967703 | 0.76% |
| 200000 | 2441214 | 6835400 | 6795109 | 0.59% |
| 250000 | 3107304 | 8700451 | 8643227 | 0.66% |
| 300000 | 3783461 | 10593691 | 10530843 | 0.59% |

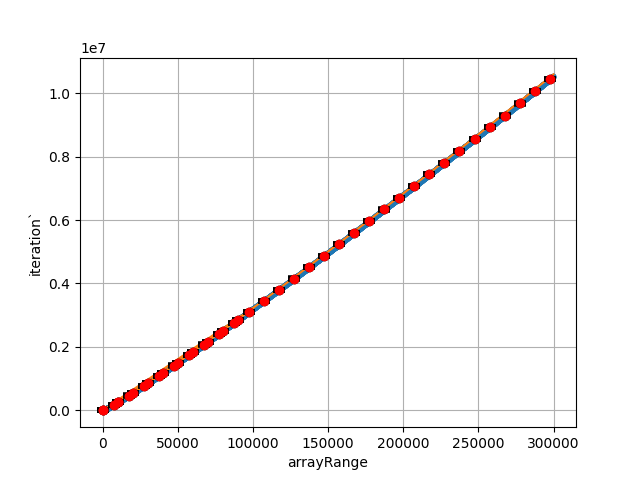
Таблица итераций:

|  |  |  |
| --- | --- | --- |
| N | Обмены | Рекурсивные вызовы |
| 50000 | 761944 | 736944 |
| 70000 | 1099311 | 1064311 |
| 90000 | 1447784 | 1402784 |
| 100000 | 1624243 | 1574243 |
| 150000 | 2521055 | 2446055 |
| 200000 | 3447930 | 3347930 |
| 300000 | 5339352 | 5189402 |

1. **Вывод:**

В ходе выполнения задания была произведена оценка трудоемкости пирамидальной сортировки, были произведены замеры количества итераций на разных размерах одномерного массива. Выяснено, что для данной выборки данных асимптотический множитель (O) равен 2.8, и трудоемкость сортировки равнаи устойчива для любых значений N.

Приложение 1. График трудоемкости.



Приложение 2. Текст программы C.

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
const int LIMIT = 100000;  
const int STEP = 100;  
const int CALL\_COUNT = 10;  
const int RAND\_LIMITS = 1000;  
  
  
void swap(int \*a, int \*b) {  
 int tmp = \*a;  
 \*a = \*b;  
 \*b = tmp;  
}  
  
void heapify(int arr[], int n, int i, int \*rec\_count, int \*swap\_count) {  
 int largest = i;  
 (\*rec\_count)++;  
 int l = 2 \* i + 1;  
 int r = 2 \* i + 2;  
  
 if (l < n && arr[l] > arr[largest])  
 largest = l;  
  
 if (r < n && arr[r] > arr[largest])  
 largest = r;  
  
 if (largest != i) {  
 (\*swap\_count)++;  
 swap(&arr[i], &arr[largest]);  
  
 heapify(arr, n, largest, rec\_count, swap\_count);  
 }  
}  
  
void heapSort(int arr[], int n, int \*rec\_count, int \*swap\_count) {  
 for (int i = n / 2 - 1; i >= 0; i--)  
 heapify(arr, n, i, rec\_count, swap\_count);  
  
 for (int i = n - 1; i >= 0; i--) {  
 swap(&arr[0], &arr[i]);  
 (\*swap\_count)++;  
 heapify(arr, i, 0, rec\_count, swap\_count);  
 }  
}  
  
void call(int len) {  
 int \*arr;  
 int rec\_count = 0, swap\_count = 0;  
 long time\_start, time\_finish;  
 arr = (int \*) malloc(len \* sizeof(int));  
 if (!arr)exit(-1);  
 FILE \*fp1;  
 FILE \*fp2;  
 if ((fp1 = fopen("write.txt", "a")) == NULL) {  
 printf("Can't open file 'write.txt'\n");  
 exit(-1);  
 }  
 if ((fp2 = fopen("write\_without\_time.txt", "a")) == NULL) {  
 printf("Can't open file 'write.txt'\n");  
 exit(-1);  
 }  
 for (int i = 0; i < len; ++i) {  
 arr[i] = rand() % RAND\_LIMITS;  
 }  
   
 time\_start = clock();  
 heapSort(arr, len, &rec\_count, &swap\_count);  
 time\_finish = clock();  
   
 fprintf(fp1, "%d %d %d %li\n", len, rec\_count, swap\_count, time\_finish - time\_start);  
 fprintf(fp2, "%d %d\n", len, rec\_count + swap\_count);  
 fclose(fp1);  
 fclose(fp2);  
 free(arr);  
}  
  
void call\_cycle(int len) {  
 for (int i = 0; i < CALL\_COUNT; ++i) {  
 call(len);  
 }  
}  
  
int main() {  
 srand(time(NULL));  
 for (int i = 10; i < LIMIT; i += STEP) {  
 call\_cycle(i);  
 }  
 return 0;  
}

Приложение 3. Текст программы Python.

import matplotlib.pyplot as plt  
import numpy  
import scipy.optimize as opt  
import numpy as np  
  
  
def func(x, c):  
 return c \* x \* np.log(x)  
  
  
def model(x):  
 return 2.8 \* x \* np.log(x)  
  
  
data = numpy.transpose(numpy.loadtxt('read.txt'))  
mean\_data = np.array([])  
std\_data = np.array([])  
arg\_data = np.array([])  
  
for i in range(0, int(len(data[0])), 10):  
 arg\_data = np.append(arg\_data, data[0][i])  
 mean\_data = np.append(mean\_data, numpy.mean(data[1][i: i + 9]))  
 std\_data = np.append(std\_data, numpy.std(data[1][i: i + 9]))  
  
popt, pcov = opt.curve\_fit(func, arg\_data, mean\_data)  
plt.plot(arg\_data, mean\_data, '.', arg\_data, model(arg\_data))  
  
bar\_arg\_data = np.array([])  
bar\_mean\_data = np.array([])  
bar\_std\_data = np.array([])  
  
for i in range(0, int(len(mean\_data)), 100):  
 bar\_arg\_data = np.append(bar\_arg\_data, arg\_data[i])  
 bar\_mean\_data = np.append(bar\_mean\_data, mean\_data[i])  
 bar\_std\_data = np.append(bar\_std\_data, std\_data[i])  
  
plt.errorbar(bar\_arg\_data, bar\_mean\_data, bar\_std\_data, fmt='o', color='r', capthick=2, elinewidth=2, capsize=3,  
 zorder=10)  
plt.errorbar(bar\_arg\_data, bar\_mean\_data, bar\_std\_data, fmt='o', color='k', capthick=4, elinewidth=4, capsize=4,  
 zorder=5)  
  
plt.xlabel('arrayRange')  
plt.ylabel('iteration`')  
plt.grid()  
plt.show()